YEAH Session #5

November 4, 2014, 6-7 PM

Nick Troccoli

YEAH Hours Schedule

Topic	Date	Time	Location
Assignment 5	Today!	Now!	Here!
Assignment 6	11/13 (Thurs)	6-7PM	Hewlett 200
Assignment 7	11/21 (Fri)	4:15-5:05PM	Hewlett 200
Final Exam	12/10 (Wed)	12:15-3:15PM	TBD

ArrayLists

- Why ArrayLists?
  - You may not know how much data you have
- ArrayLists can grow as big as you need
- Can check if something is in it with .contains()
- Can only store objects (no primitives!)
  - Need “Integer” instead of “int” for storing integers, etc.

```
ArrayList<TYPE> list =
  new ArrayList<TYPE>();
```

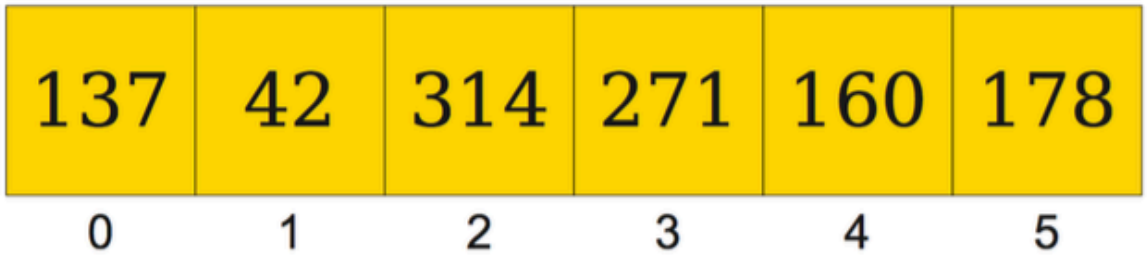
ArrayList

`boolean add(<T> element)` Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.
`void add(int index, <T> element)` Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.
`<T> remove(int index)` Removes the element at the specified position and returns that value.
`boolean remove(<T> element)` Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.
`void clear()` Removes all elements from the **`ArrayList`**.
`int size()` Returns the number of elements in the **`ArrayList`**.
`<T> get(int index)` Returns the object at the specified index.
`<T> set(int index, <T> value)` Sets the element at the specified index to the new value and returns the old value.
`int indexOf(<T> value)` Returns the index of the first occurrence of the specified value, or −1 if it does not appear.
`boolean contains(<T> value)` Returns **`true`** if the **`ArrayList`** contains the specified value.
`boolean isEmpty()` Returns **`true`** if the **`ArrayList`** contains no elements.

Arrays

- Why Arrays?
  - Arrays are great for representing a fixed-sized list
- Store data at difference indices in the array, and look up data by index
- Can store any type of data (objects & primitives)

Arrays



- An array stores a **sequence** of multiple objects.
  - Can access objects by index using [].
- All stored objects have the same type.
  - You get to choose the type!
- Can store *any* type, even primitive types.
- Size is fixed; cannot grow once created.

Basic Array Operations

- To create a new array, specify the type of the array and the size in the call to **`new`**:

```
Type[] arr = new Type[size]
```

- To access an element of the array, use the square brackets to choose the index:

```
arr[index]
```

- To read the length of an array, you can read the **`length`** field:

```
arr.length
```

2D Arrays (Grids)

```
Type[][] a = new Type[rows][cols];
```

Interpreting Multidimensional Arrays

- There are two main ways of intuiting a multidimensional array.
- **As a 2D Grid:**
  - Looking up ***arr*`[`*row*`]` `[`*col*`]` selects the element in the array at position (***row***, ***col***).
- **As an array of arrays:**
  - Looking up ***arr*`[`*row*`]` gives back a one-dimensional consisting of the columns in row ***row***.

Iterating through a 2-D array

```
Type[][] arr = /* … */
for (int row = 0; row < arr.length; row++) {
  for (int col = 0; col < arr[row].length; col++) {
    /* … access arr[row][col] … */
  }
}
```

2D Array Example

```
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
   for (int col = 0; col < arr[row].length; col++) {
      arr[row][col] = row + col;
   }
}
```

	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	5
2	2	3	4	5	6
3	3	4	5	6	7

Yahtzee!

- Due at 3:15PM on Wednesday, Nov. 12
- Graphics already implemented for you!
- Practice with arrays
- YahtzeeDemo – working demo in assignment folder (double-click to play)
- YahtzeeMagicStub.checkCategory() – provided for you for testing (eventually need to check category yourself!)

DEMO!

Game Flow

1. Roll Dice 1st time
2. Select a set of dice to reroll (if any) and reroll
3. Repeat step 2
4. Choose a category to score in (***make sure they haven't already used it***!)

Use YahtzeeDisplay!

YahtzeeDisplay

- Graphics are taken care of for you
- Manipulate onscreen graphics via your YahtzeeDisplay instance variable (display.__)
- Methods on YahtzeeDisplay (from handout):
  - waitForPlayerToClickToRoll()
  - displayDice()
  - waitForPlayerToSelectDice()
  - isDieSelected()
  - waitForPlayerToSelectCategory()
  - updateScorecard()
  - printMessage()
- Player indices start at 1!!

Constants

```
/** The number of dice in the game */
   public static final int N_DICE = 5;

/** The maximum number of players */
   public static final int MAX_PLAYERS = 4;

/** The total number of categories */
   public static final int N_CATEGORIES = 17;

/** The number of categories in which the player ca
   public static final int N_SCORING_CATEGORIES = 1

/* The constants that specify categories on the sco
   public static final int ONES = 1;
   public static final int TWOS = 2;
   public static final int THREES = 3;
   public static final int FOURS = 4;
   public static final int FIVES = 5;
   public static final int SIXES = 6;
   public static final int UPPER_SCORE = 7;
   public static final int UPPER_BONUS = 8;
   public static final int THREE_OF_A_KIND = 9;
   public static final int FOUR_OF_A_KIND = 10;
   public static final int FULL_HOUSE = 11;
   public static final int SMALL_STRAIGHT = 12;
   public static final int LARGE_STRAIGHT = 13;
   public static final int YAHTZEE = 14;
   public static final int CHANCE = 15;
   public static final int LOWER_SCORE = 16;
   public static final int TOTAL = 17;
```

Calculating Scores

- Given a set of dice, calculate the score for the chosen category
- 1s, 2s, 3s, full house, small straight…
- Use YahtzeeMagicStub initially/for testing, but don't use it for your final submission!

```
boolean matches =
YahtzeeMagicStub.checkCategory(dice, YAHTZEE);
```

Calculating Scores

- Any roll is valid for 1s, 2s, 3s, 4s, 5s, 6s, and chance
- 3 Of a Kind, 4 Of a Kind, Yahtzee, Full House, Straights -> not all rolls valid (score = 0 if roll doesn't fit category!)
- Update total score each time!
- When checking if roll fits category, think about dice value *frequencies* (e.g. what is 3 of a kind with respect to dice value frequencies?)

Game End

- Tally up Upper Bonus, Upper Score, Lower Score, Final Total
- Report winner!

Arrays Galore!

- Dice (N_DICE)
- Players (array of player names given to you in starter code as instance variable)
- Scorecard for all players (2d array representing scorecard)

Testing Tips

- Use System.out.println() to print testing messages to the Eclipse console (can't use println() because Yahtzee isn't a ConsoleProgram!)
- Hardcode dice array so you always control what the dice rolls are (great for testing logic for scoring categories)
- Think about dice value *frequencies* when checking if a roll fits a given category

Final Tips

- Follow the specifications carefully
- Extensions!
- Comment!
- Go to the LaIR if you get stuck
- **Incorporate IG feedback!**

- Have fun!